Classification

The classification tool's functions are divided over tree different processes that can be run independently from each other. This means that you can run all processes in one go, or run only the data splitting once, and run the training and testing processes repeatedly later. However, the training process cannot be run if the dataset has not yet been split into train, test and validation set. Similarly, the testing process cannot be run without a trained model etc. Below is the description of each of these processes. Which processes are executed can be set from the settings file using the lines below.

```
execute dataset splitting process = true
execute train process = true
execute test process = true
execute classification process = true
execute_output_features_csv_process = true
```

Data splitting

Splitting the data can be done by executing the data splitting process. This process loads annotated measurements from the source that is defined in the settings file. The definition of a single measurement in this context is the combination of a single x, y and z direction of the accelerometer, together with its speed as measured by the gps. As a source, a mat-file can be selected. This mat-file should be located in the data folder.

```
annotated_measurement_source_paths = 210911_meeuw_alldata_reformatted.mat
```

Multiple mat-files can be selected using the same setting. In this case a comma separated list of file names can be given. Make sure not to use line breaks within the list. Below is an example of loading 2 source files.

```
annotated_measurement_source_paths = Anot6020_4343.mat, Anot6020_4352.mat
```

It is recommended that measurements containing NaN (not a number) values are filtered out. This can be done using the setting below.

```
remove_measurements_containing_nan = true
```

Classification can only be done on a series of measurements or a segment. Therefore a number of consecutive measurements are grouped into a single segment. The number of measurements in a single segment can be defined the setting below.

```
measurement_segment_size = 20
```

Only a homogeneous segment can be created. This means that there will never be created a segment that contains measurements with various time stamps, device ids or different labels. If there are too few consecutive homogeneous measurements to form a segment, these measurements are all discarded. If a device recorded many measurements at a given time, multiple segments can be created with the same device and time stamp. In some situations, these instances cannot be regarded as independent. To prevent multiple segments from being created from data of a single device and time stamp, see the line below.

```
segments_must_have_unique_id_timestamp_combination = true
```

The labels, their description, and their associated colors used for visualization, are defined in a schema file. Note that, the schema file should be located in the data folder of the software. From the settings file, the used schema can be set. The lines below indicate that the file at location */data/schemaGull.txt* is to be used.

```
label_schema_file_path = schemaGull.txt
```

Below is a typical schema file, defining the details of the classes 1 through 9.

```
1 stand 1.00 0.00 0.00
2 flap 0.75 0.00 0.75
3 soar 0.00 0.00 1.00
4 walk 0.00 0.50 0.00
5 sit 0.50 0.50 0.50
6 XflapL 1.00 0.80 1.00
7 float 1.00 1.00 0.00
8 XflapS 0.00 1.00 1.00
9 other 0.00 1.00 0.00
```

It can be chosen to merge several classes. This can be done using a remapping schema. Such a schema can be defined in a file. In the settings file the remapping schema file name can be set. As with the schema file, the schema remapping file should be located in the data folder.

```
label_ids_must_be_remapped = true
label_schema_remapping_path = schemaGullRemap.txt
```

The schema remapping file is a space delimited text file. The file below would cause original labels 1 and 2 to be merged in to a new label 1, with the description 'Feeding'. Original label 3 is given the new label 2 with description 'Walk' etc.

```
1 Feeding 1,2
2 Walk 3
3 Stand 4
4 Preen 5
5 Fly 6,7,8
```

```
6 Other 9
```

The resulting segments are then divided over 3 sets, a train set, a test set, and a validation set. The train set is a set that is used for training the model, also known as classifier. Training a model could for instance mean generating a decision tree. The test set is for testing the performance of the trained model. Trying out different training methods with various parameters cause several trained models that vary in performance on the test set. By picking out the best performing model, the model is somewhat optimized to the test set. To objectively test the performance of the final version of a trained model, a validation set is used. The segments can be split over these data sets according to ratios defined in the settings file. The lines below define a 50%, 25%, 25% distribution.

```
dataset_split_train_ratio = 0.5
dataset_split_test_ratio = 0.25
dataset_split_validation_ratio = 0.25
```

The lines below are equivalent.

```
dataset_split_train_ratio = 50
dataset_split_test_ratio = 25
dataset_split_validation_ratio = 25
```

Often, the data is not equally divided over the available classes. This could result in the poor performance of classes of which there is less data available. In some extreme cases, the training process can even choose to ignore a class completely. Therefore a fixed number of segments for each class can be defined. If this method is used, the defined number of segments are taken from the train set. The rest of the train set is discarded. The lines below will result in sampling 25 segments of each class, 1 through 5, to train on. Note that, when using a schema remapping, the new, remapped labels are used in this setting.

```
train_on_fixed_class_numbers = false
train_instances_per_class = 1:25, 2:25, 3:25, 4:25, 5:25
```

After this, the train, test and validation sets are saved in json format in the data folder. The file names under which the data sets are saved are set in the settings file.

```
train_set_file_path = train_set.json
test_set_file_path = test_set.json
validation_set_file_path = validation_set.json
```

Training

During the training process, the train set is loaded from the data folder. The name of the file from which train instances should be loaded is defined in the settings file.

```
train_set_file_path = train_set.json
```

For each of the segments in the train set features are calculated. Based on these features, a model is trained. From a fixed set of features, a selection can be picked that is used for training. Note that the trained model can only classify instances that have the exact same features. What features are going to be used can be set in the settings file. The line below causes only the means and their standard deviations of each dimension to be used as features.

```
extract_features = mean_x, mean_y, mean_z, std_x, std_y, std_z
```

Besides the standard features, user-defined features can be added. These can be defined in a text file. The location of the text file, in the data folder, can be set with the following line.

```
custom_feature_extractor_file_path = custom_features.txt
```

The format of the file containing custom feature definition is semicolon seperated. Below is an example of such a file in which 2 features are defined.

```
myfeature; x0*y0*z0 + x1*y1*z1 + x2*y2*z2 + x3*y3*z3
someotherfeature; x0+x1+x2+x3 - 4 * meanx
```

The first column contains the name of the feature, and the second column contains the expression defining the feature. For interpreting the expression, the exp4j library is used. Note that variable names x0, x1 etc. refer to the different elements of x of the segment. The variable names xmean and stdx refer to the mean and standard deviations of x. The complete list of variable names is below.

```
x0
x1
x2
...
y0
y1
y2
...
z0
z1
z2
...
meanx
stdx
meany
stdy
meanz
```

```
stdz
```

Note that while defining new features, the number of elements have to be taken into account. Using z30, for example, on a segment of only 20 measurements will result in an error.

A model is trained on the train set using a machine learning algorithm. Which algorithm should be used, is set in the settings file. To use the C4.5 treelearning algorithm use the line below. Note that j48 is the name of a JAVA implementation of the C4.5.

```
machine_learning_algorithm = j48
```

The line below causes the Random Forest algorithm to be used.

```
machine_learning_algorithm = random_forest
```

Most machine algorithms have specific settings, or parameters, that can be tuned. For instance, the number of trees in the Random Forest can be defined.

```
random_forest_number_of_trees = 2000
```

After training, the resulting trained model, also known as classifier, is saved in the data folder under the file name that is defined in the settings file.

```
classifier_path = classifier.cls
```

If the trained model is a decision tree, its structure is also saved in the job folder under "treegraph.json". By opening "treevisualization.html" in the same folder, the structure of the tree can be visualized. More complex tree structures can be inspected further by collapsing intermediate nodes.

Testing

Testing is done to evaluate the performance of a trained model. The trained model that is used to evaluate can be set from the settings file. The file containing the trained model should be located in the data folder.

```
classifier_path = classifier.cls
```

The file, within the data folder, containing the test set is also set from the settings file.

```
test_set_file_path = test_set.json
```

For each of the segments in the test set features are calculated. Note that the same features need to be used for testing as were used for training the model. For further explanation of features and how to set which ones are use, see the Training section.

After testing, a test report is generated, containing some statistics, including an error rate and a confusion matrix. This report is saved in the job folder under the file name "test_report.txt". Misclassified instances of the test set are saved in json format in the job folder under "misclassifications.json". By opening "misclassifications.html", visualizations of the misclassifications can be inspected together with their predicted and actual label and feature values.

Classification

During the classification process, unannotated data can be labeled using a trained model. From which file the model is to be loaded can be set from the settings file. The file should be located in the data folder.

```
classifier_path = classifier.cls
```

Unannotated data is loaded from a mat-file and segmented the same way as in the datasplitting process. Features are calculated for each segment the same way as described in the Training section. The file containing the unannotated measurements should be in the data folder. Its file name can be set from the settings file.

```
unannotated_measurement_source_paths = 10000_unannotated_538.mat
```

The file should contain a similar struct to the one below.

```
outputStruct =

nOfSamples: 60
sampleID: [1x60 double]
year: [1x60 double]
month: [1x60 double]
day: [1x60 double]
hour: [1x60 double]
min: [1x60 double]
sec: [60x1 double]
accX: {60x1 cell}
accY: {60x1 cell}
accZ: {60x1 cell}
accP: []
accT: []
```

```
tags: {1x60 cell}
annotations: [33x6 double]
gpsSpd: [60x1 double]
```

After classifying, all segments have been assigned a label by the model. The results are saved in an csv file called "classifications.csv" in the job folder. Each row in this file describes a single segment and its classification. The columns in this file are described below.

```
device_info_serial  The device id
date_time           The time at which the device started recording accelerometer data
first_index         The index of the first measurement of the segment
class_id            The id of the assigned class
class_name          The name of the assigned class
class_red           The amount of red in the associated color of the class (for visualisations only)
class_green         The amount of green in the associated color of the class (for visualisations only)
class_blue          The amount of blue in the associated color of the class (for visualisations only)
longitude           The longitude measured by the GPS when the accelerometer started recoring
latitude            The latitude measured by the GPS when the accelerometer started recoring
altitude            The altitude measured by the GPS when the accelerometer started recoring
gpsspeed            The speed measured directly by the GPS when the accelerometer started recoring
```

Below is an example of an output of the classification process.

```
device_info_serial,date_time,first_index,class_id,class_name,class_red,class_green,class_blue,longitude,latitude,altitude,gpsspeed
538,2011-06-08T08:05:31.000Z,0,3,soar,0.0,0.0,1.0,4.4429206,52.7374668,-4.0,0.8106154424176607
538,2011-06-08T08:15:07.000Z,0,7,float,1.0,1.0,0.0,4.4449913,52.7411828,-2.0,0.6645039573775211
538,2011-06-08T08:19:55.000Z,0,7,float,1.0,1.0,0.0,4.4460503,52.7427955,-5.0,0.6157412031979498
```

Often, we are interested in a label for each device-timestamp combination as opposed to for each segment. Such device-timestamp combination can have multiple segments with possibly different labels. To cope with this, every first segment with a unique device id - timestamp combination is used as leading for the complete device-timestamp combination. If no segments exist for an id-timestamp combination, no label is given.

The output csv file from the classification process can be used in other applications, like Matlab, for further analysis. The file can also be uploaded to csv update section of the Annotation tool.

Feature output

This process is special process only meant to save statistics to be analysed within Matlab or Excel for example. In this process, train, test and validation sets are loaded from the locations set in the settings file. For each segment, device, timestamp, longitude, latitude, altitude, annotated label, and list of features is output on a single row. The output is saved under "featurescomplete.csv" in the job folder.

Settings

In the sections above, all settings in the settings file are discussed. Because only a selection of the above processes can be selected, setting all possible settings is sometimes counter-intuitive. When only the data splitting process is selected, it doesn't make sense to have to set a *classifier_path* for instance, as this setting is obviously not needed to run that particular process.

It was chosen however to check if every possible setting has been set before executing. This ensures that no processes have to be stopped halfway because of a missing setting.